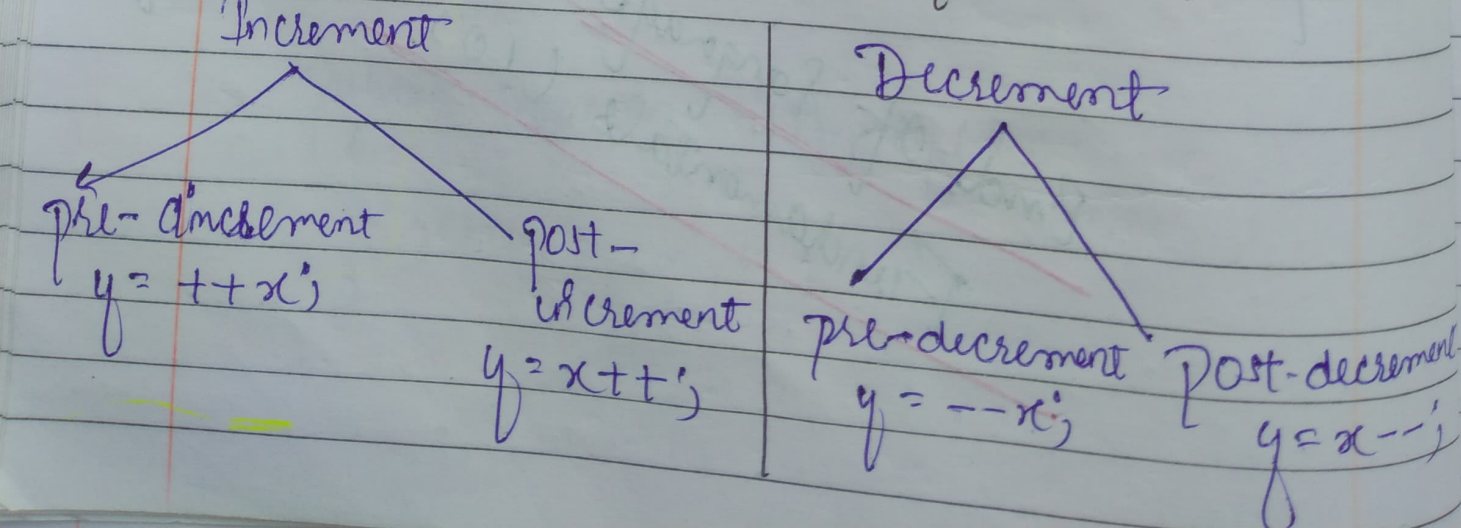
Lecture 17 :- 6/23/22 12:20 PM

CLASSTIME Pg. No. Date / /

* Operators & Assignments :-

1) Increment and Decrement operators.
2) Arithmetic operators
3) String Concatenation operators
4) Relational operators.
5) equality operators.
6) instanceOf operators
7) bitwise operators.
8) short circuit operators.
9) type cast operators
10) assignment operators
11) Conditional operator.
12) new operator.
13) [] operator
14) operator precedence
15) Evaluation order of operands.
16) new Vs newInstance().
17) instanceof Vs isInstance()
18) ClassNotFoundException Vs NoClassDefFoundError.

(1) Increment & Decrement operators :-



CLASSTIME Pg. No.
Date / /

Ex:-

Expression	Initial value of x	value of y	final value of x.
y = ++x;	10	11	11
y = x++;	10	10	11
y = --x;	10	9	9
y = x--;	10	10	9.

(1) * We can apply increment & decrement operators only for variables but not for constant values. If we are trying to apply/ing for constant values then we will get CTE.

Ex:
```
int x=10;
int y = ++x;
sopln(y);
```

```
int x=10;
int y = ++10;
sopln(y);
```

CE: unexpected type found: value. req: variable.

(2) * Nesting of Increment & Decrement operators not allowed.

Ex:
```
int x = 10;
int y = ++(++x);
sopln(y);
```

CE: Unexpected type req: variable found: value.

(3) For Final variables we can't apply increment & dicrement operators.

```
final int x = 10;
x++;
sopln(x);
```

CE: cannot assign a value to final variable x.

```
final int x = 10;
x = 11;
sopln(x);
```

CE: cannot assign a value to final variable x.

(4)

1)
```
int x = 10;
x++;
sopln(x); 11
```

```
char ch = 'a';
ch++;
sopln(ch); b.
```

2)
```
double d = 10.5;
d++;
sopln(d); 11.5
```

```
boolean b = true;
b++;
sopln(b);
```

CE: {operator ++ can not be applied to boolean.}

CLASSTIME Pg. No. Date / /

We can apply increment & decrement operators for every primitive types except boolean.

(5) * Difference B/w b++ and b = b+1 :-

If we apply any arithmetic operator B/w two variables a & b the result type is always max of (int, type of a, type of b).

max (int, type of a, type of b)

max (int, byte, byte).

Ex: byte a = 10;
byte b = 20;
byte c = (a+b); ⇒ max (int, byte, byte). result is int.
sopln (c);

CE: possible loss of precision
found: int
required: byte

How we can handle this CTE?

byte c = (byte) (a+b);
sopln (c); 30

We solve this by using type casting.

Ex 2: byte b = 10;
b = b+1;
sopln (b);

CE: PLP
found: int
req: byte

result type first arg.
max (int, byte, int)
int ⇐ type of 2nd argument.
b = (byte) (b+1)

old ka hai

```
byte b=10;
b++;
sopln(b); //
```

A `b=(byte)(b+1)` — ✓ solve by using type Casting

But, in the case of increment & decrement operator, internal typecasting will be performed automatically.

Ex:- `byte b=10;`

```
b++;
sopln(b);
```

`b=(byte)(b+1);`

`b++;` ⇒ `b=(typeofb)(b+1)`

7:16PM(2) Arithmetic Operators :- (+, -, *, /, %(.))

1) 6/25/22 If we apply any arithmetic operator on two variables a & b the result type is always -

max(int, type of a, type of b).

byte + byte = int
byte + short = int
short + short = int
byte + long = long.
long + double = double
float + long = float
char + char = int → eg: sopln('a' + 'b');
97 + 98 ⇒ 195 int.

byte → short → int → long → float → double
char →

min Left → Right Max

Chart of

char + double = double. → sopln('a' + 0.89); 98.89
97 + 0.09

Infinity :- In integral arithmetic (byte, short, int, long) there is no way to represent infinity. Hence, if infinity is a result we will get AE: in integral arithmetic. eg:-

sopln(10/0); RE: AE: / by zero.

But in floating point arithmetic (float & double) there is a way to represent infinity for this Float & Double classes contains the following two contants -

POSITIVE_INFINITY &
NEGATIVE_INFINITY. Hence, even though result is infinity we want get any arithmetic exception in floating point arithmetic.

eg:- sopln(10/0.0); Infinity
sopln(-10.0/0); -Infinity.

NaN:- (Not a Number). In integral arithmetic byte, short, int, long there is no way to represent undefined result hence if the result is undefined we will get RE: ArithmeticException: / by zero. eg:-

sop(0/0); RE: AE: / by zero.

But in floating point arithmetic (float & double) there is a way to represent undefined results for this Float & Double classes contains NaN content. Hence if the result is undefine

• In general at lang. Level Java won't provide operator overloading.

we want get any AE in° floating point arithmetic.

```
sopln(0.0/0);    NaN.
sopln(+0.0/0);
sopln(-0/0.0);   NaN.
```

Summary.

```
sopln(10/0);     RE: AE: / by zero
sopln(10/0.0);   Infinity
sopln(0/0);      RE: AE: / by zero
sopln(0.0/0);    NaN.
```

Note: for any x value including NaN the following expressions returns false.

1) $x < NaN$
   $x <= NaN$
   $x > NaN$
   $x >= NaN$
   $x == NaN$
   $\Rightarrow$ false.

2) $x != NaN \Rightarrow$ true

for any x value including NaN the following expressions returns true.

$x != NaN \Rightarrow$ true.

Eg:

```
sopln(10 < Float.NaN);    false
sopln(10 <= Float.NaN);   false
sopln(10 > Float.NaN);    false
sopln(10 >= Float.NaN);   false
sopln(10 == Float.NaN);   false
sopln(Float.NaN == Float.NaN);   false
```

sopln (10/0 = Float.NaN); true
sopln (Float.NaN != Float.NaN); true.

* Arithmetic Exception :-

1. It is Run-time exception but not Compile Time error.
2. It is possible only in integral arithmetic but not in floating point arithmetic
3. The only operators which cause AE are / and %.
   - / → division
   - % → Modulo

06/26/22 2:20. **Lecture 18 :- (3) String concatenation operator (+)**

The only overloaded operator in Java is (+) operator. Sometimes it acts as arithmetic addition operators and sometimes it acts string concatenation operator.

If atleast One argument is String type then (+) operator acts as Concatination operator and if both arguments are number type then (+) operator acts as arithmetic addition operator.

eg :-
```
String a = "jahad";
int b=10, c=20, d=10;
sopln(a+b+c+d);   jahad102030
sopln(b+c+d+a);   60jahad
sopln(b+c+a+d);   30jahad10
sopln(b+a+c+d);   10jahad2030.
```

CLASSTIME Pg. No.
Date / /

Analysis

a + b + c + d

"fahad10" + c + d

"fahad1020" + d

fahad102030. Ans

If expression have same operator then order of precidence are always from Left to → right.

Ex-2: Consider the following declarations :-

String a = "durga";
int b = 10, c = 20, d = 30;

Which of the following expressions are valid.

✗ ① a = b + c + d; → CE: incompatible found: int type req: T. & String

✓ ② a = a + b + c;

✗ ③ b = a + c + d; → CE: incompatible types found: T. & String req: int

✓ ④ b = b + c + d;

(4) * Relational operators :- (<, <=, >, >=) :-

(1) We can apply relational operator for every primitive types except boolean.

eg:-

sopln (10 < 20); true
sopln ('a' < 10); false
sopln ('a' < 97.6); true

sopln ('a' > 'A'); true (∵ 97 > 65)
(97, 65)
sopln (true > false); → CE: operator > cannot be applied to boolean, boolean

2) * We cant apply relational operators for object types.

eg: sopln ("durga123" > "durga");

→ CE: operator > Cannot be applied to Java.l.String, J.l.String.

3) * Nesting of relational operators is not allowed otherwise we will get C.T.E.

sopln (10 < 20 < 30)

↓

true < 30

operator < cannot be applied to boolean, int.

4) * Equality operators (==, !=).

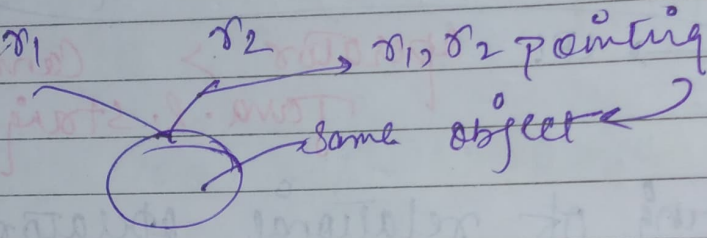
We can apply equality operators for every primitive types including boolean type also.

sopln (10 == 20); false
sopln ('a' == 'b'); false
sopln ('a' == 97.0); true
sopln (false == false); true.

We can apply equality operators for Object types also:

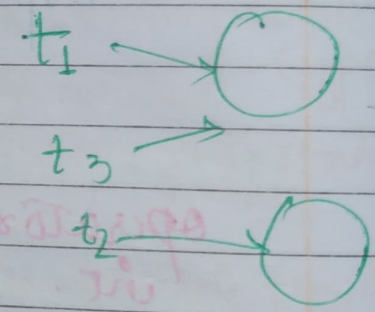
for object references r_1, r_2.

$r_1 == r_2$ ⇒ returns true iff both references pointing to the same object (reference comparision or address comparision).



eg1:-
```
Thread t1 = new Thread();
Thread t2 = new Thread();
Thread t3 = t1;

Sopln (t1 == t2);
Sopln (t1 == t3);
```



If we apply equality operators for Object types then compulsory there should be some relation b/w argument types (either child to parent, or parent to child, or same type), otherwise we will get C.T.E saying incomparable types: Java.lang.String and Java.lang.Thread.

```
Thread t = new Thread();
Object o = new Object();
```

String s = new String ("durga");

sopln (t == 0); false
sopln (0 == s); false
sopln (s == t); → CE: incomparable types; Found: String and Thread.

t → ○
0 → ○
s → "durga"

In general we can use == operator for reference comparision (address comparision) and .equals() method for content comparision.

Eg:- String s1 = new String ("durga");
String s2 = new String ("durga");
sopln (s1 == s2); false
sopln (s1.equals(s2)); true.

Difference b/w == operator & .equals() method.

s1 → "durga"
s2 → "durga"

Note:- for any object reference 'r':
r == null is always ⇒ false.

r == null ⇏ false

But null == null is always true.

String s = new String ("durga");
sopln (s == null); false

String s = null;
sopln (s == null);
true.

CLASSTIME Pg. No.
Date / /

sopln (null = null); true.

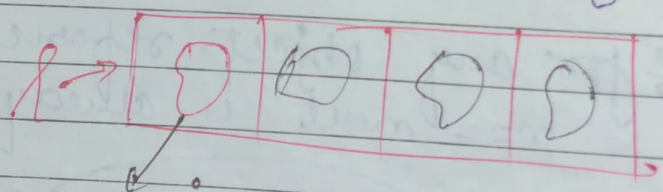
6/26/22 21:48 PM

Lecture 19 :-

(6) Instanceof operator :-

We can use instanceof operator to check whether the given object is of particular type or not.

```
object o = l.get(0);   // zero
if (o instanceof Student)
    Student s = (Student)o;   // zero
    // perform Student specific functionality;
else if (o instanceof Customer)
{
    Customer c = (Customer)o;   // zero
    // perform Customer specific functionality
}
```



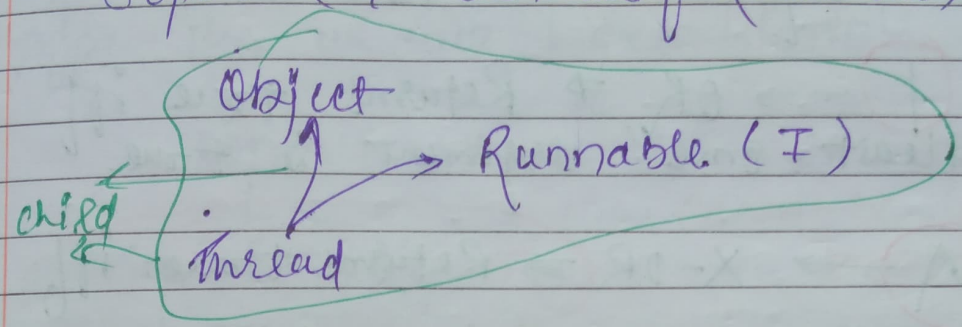
* Syntax :-

x instance X

x → object reference

X → class / interface name

```
Thread t=new Thread();
sopln(t instanceof Thread); true
sopln(t instanceof Object); true
sopln(t instanceof Runnable); true.
```



To use instanceof operator compulsory there should be some relation b/w arguments types either child to parent, parent to child or same type, otherwise we will get CTE saying :

Eg:
② sopln(t instanceof String);
CE: inconvertible types
found: j.l.Thread
required: j.l.String.

① Thread t = new Thread();

Note: for any class or interface 'X'

null instanceof X → is always false.

sopln(null instanceof Thread); false
sopln(null instanceof Runnable); false.

CLASSTIME Pg. No. Date, Error

⑦ Bitwise operators (&, |, ^) → inclusive OR
and OR

(&) → AND ⇒ Returns true iff both arguments are true.

(|) → OR ⇒ Returns true iff atleast one argument is true.

Cap symbol → (^) → X-OR ⇒ Returns true. iff both arguments are different.

Eg:- sopln (true & false); false
sopln (true | false); true
sopln (true ^ false); true.

We can apply these operators for integral types also.

```
sopln (4 & 5);
sopln (4 | 5);
sopln (4 ^ 5);
```

4 → 100
5 → 101

100
& 101
100

100
| 101
101

100
^ 101
001

Bitwise Complement Operator :- (~) :- tild symbol

We can apply this operator only for integral types but not for boolean types. If we are trying to apply for boolean type then we will get C.T.E.

```
soplu ( ~true )';  → CE: Operator ~ cannot be applied to boolean.
soplu (~4)';  --5
```

→ Explanation :- 4 is int type and have 8 byte i - size and

1 byte = 8 bit
so 32 bit total

Memory level Representation

4 ⇒ 0 0000 0100

(+) means, MSB, Sign bit, value (using 31 bit)

~4 ⇒ 1 111 1011

-ve, value

Explanation :- we have to find 2's complement because -ve symbol.

1's compl 1st step ⇒ 000 . . . 0100

2's compl → 1's compl + 1

00 . . . . 0101 → 5

Ans -5

32-bit complement we have to perform not for 3-4 bit.

Note: The Most Significant bit act as sign bit.

→ 0 → means +ve number
→ 1 → means −ve number.

* +ve numbers will be represented directly in a memory whereas −ve number will be represented indirectly in a memory in 2's complement form.

Case Study:

4 | 5 ⇒ 000.....0100 → 4
4 | 5   000.....0101 → 5
4 | 5   000.....100 → 4

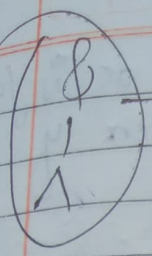
* Boolean Complement Operator: (!)

We can apply this operator only for boolean types but not integral types.

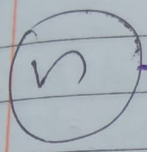
eg: sopln(!4); → CE: Operator ! Cannot be applied to int.

→ sopln(!false); true ✓

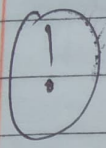
CLASSTIME Pg. No.
Date / /



→ applicable for both boolean & integral types.



→ Applicable for only integral types but not for boolean type.



→ applicable only for boolean but not for integral types

6/26/22 Lecture 20 :- (8) Short Circuit Operators :- (&&, ||)

These are exactly same as bitwise operators (&, |.) except the following differences.

& , |	&& , ||
1. Both arguments should be evaluated always.	1) Second argument evaluation is optional.
2. Relatively performance is low.	2) Relatively performance is high.
3. Applicable for both boolean & integral type	3) applicable only for boolean but not for integral types.

Note :- (1) x && y ⇒ y will be evaluated if x is true
i.e if x is false then y won't be evaluated.

(c) x||y → y will be evaluated → if x is false i.e if x is true then y won't be evaluated.

eg:-1 int x=10, y=15;
if (++x <10 & ++y >15)
{
x++;
}
else
{
y++;
}
System.out.println(x+"..."+y);

O/P:-

	x	y
&	11	17
&&	11	16
|	12	16
||	12	16

eg2:- int x=10;
if (++x <10 && (x%0 >10))
{
System.out.println("Hello");
}
else
{
System.out.println("Hi");
}

CLASSTIME Pg. No.
Date / /

O/P :
① CE
② RE: AE: / by zero
③ Hello
④ Hi

* if && — replace with &, then ans is ② we will get RE: AE: / By zero.

⑤ Type cast operator :

① Implicit Type-casting ② Explicit Type-casting

There are two types of typecasting :
① Implicit-TC
② Explicit-TC

* Implicit Type Casting :

(1) Compiler is responsible to perform implicit type Casting.
(2) Whenever we are assigning smaller datatype value to bigger datatype variable implicit T.C will be perform.
(3) It is also byas widening or upcasting.
(4) There is no loss of information in this Type Casting.

The following are various possible conversions where implicit TC will be performed.

byte → short → int → long → float → double
char → int → long → float → double

eg 1: int x = 'a'; → compiler converts char to int automat. by Imp. sout T.C
sop(x); = 97

eg 2: double d = 10; → Comp. Const int to doubl automat. by Implict. T.C.
sopln(d); 10.0.

(2) Explicit Type casting :-

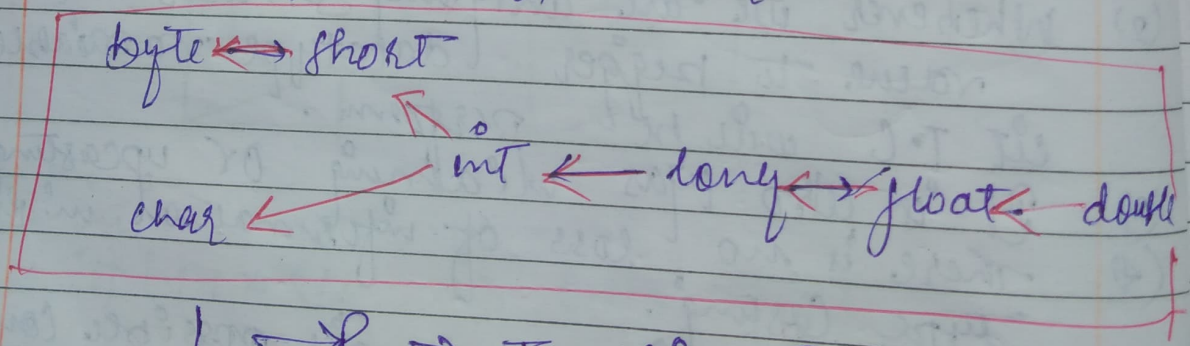
(1) programer is responsible to perform explicit T.C.

(2) Whenever we are assigning bigger D.T value to smaller D.T variable then explicit T.C is/will be required.

(3) It is also nyas narrowing or down-casting.

(4) There may be a chance of loss of information with this T.C.

The following are various possibilities where explicit type casting is required.



L → R ⇒ Implicit Type-Casting
R → L ⇒ Explicit Type-Casting.

CLASSTIME Pg. No.
Date / /

Eg: `int x = 130;`
`byte b = x;` → CE: Possible loss of precision
found: int
required: byte.

`byte b = (byte)x;` → ✓

`sop(b);` → -126 ✓

→ Whenever we are assigning bigger D.T datatype value to smaller D.T variable by implicit type-casting the MSB will be lost. We have to consider only LSB (least significant bits).

```
int x = 130;
byte b = byte(x);
sop(b) = -126.
```

int x = 130 ⇒ 0000 010000010

byte b = (byte) x ⇒ 1 000 0010

-ve

2's complement.

Consider 8-bit only due to byte size.

```
2 | 130
2 |  65 - 0
2 |  32 - 1
2 |  16 - 0
2 |   8 - 0
2 |   4 - 0
2 |   2 - 0
  |   1 - 0
```

1 1 1 1 1 0 1
+ 1
1 1 1 1 1 1 1 0
$2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

⇒ 64 + 32 + 16 + 8 + 4 + 2 + 0

⇒ 126

Eg2:- int x=150;
short s = (short)x;
sopln(s); → 150

byte b=(byte)x;
sopln(b); = -106

2 | 150
2 | 75 - 0
2 | 37 - 1
2 | 18 - 1
2 | 9 - 0
2 | 4 - 1
2 | 2 - 0
1 - 0

int x=150; ⇒ 000 ... 0|00|0110

short s=(short)x; ⇒ 0 000 ... 0|00|0110
+ve No

byte b=(byte)x; ⇒ 1 00|0|1|1 0
-ve

1 1 0 | 0 0 | 1
+ 1
1 1 0 | 0 | 1 0

$+2^4X0+2^3X1+2^2X0+2^1X1+2^0X0$ ← $2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$
2^6X1+2^5X1
⇒ 64+32+0+8+0+2+0
⇒ 106

CLASSTIME Pg. No.
Date / /

Pgm: If we assign floating Point values to the integral types by explicit typecasting the digits after the decimal point will be lost.

```
Ex: double d = 130.456;
    int x = (int)d;

    System.out.println(x); (130)

    byte b = (byte)d;
    System.out.println(b); (-126)
```

6/27/22 1:9 PM
6/28/22 5:38 PM

Lecture 2 | 6/5 (16) Assignment operators

There are 3 types of assignment operators:
(1) Simple assignment
(2) chained "
(3) Compound "

1) → Simple :- `int x=10;`

2) → chained assignment :-

```
int a,b,c,d;
a=b=c=d=20;
System.out.println(a+"..."+b+"..."+c+"..."+d);
                   20      20      20      20
```

We can't perform chained assignment directly at the time of declaration.

eg:-

```
(int a)=b=c=d=20;
```

(E! cannot find symbol
Symbol: variable b
loc: class Test
)

int b, c, d;
int a=b=c=d=20; ✓

* Compound assignment operators:

eg: int a = 10;
a += 20; → a = a+20;
sopln(a); ⇒ 30 ✓

Sometimes assignment operator merged with some other operator such type of assignment operators are called Compound assignment operators.

The following are all possible compound assignment operators in Java.

+=	&=	>>= right shift operator
-=	|=	>>>= Unsigned right shift op
*=	^=	<<= left shift operator.
/=		
%=		

(Total = 11)

In the case of compound assignment operators internal T.C will be performed automatically.

byte b = 10; → max(int, byte, int)
b = b+1; = int
sop(b); → CE: PLP found: int required: byte.

CLASSTIME Pg. No.
Date / /

```
byte b = 10;
b++;
sopln(b); 11 ✓
```

```
b = (byte)(b+1);
```

internal T.C is performing

```
byte b=10;
b += 1;
sopln(b); 11
```

b = (byte)(b+1);

In the case of compound assignment operator internal T.C performed automatically.

→
```
byte b = 127;
b += 3;
sopln(b); -126 ✓
```

```
int a, b, c, d;
a = b = c = d = 20;
a += b -= c *= d /= 2;
sopln(a+"..."+b+"..."+c+"..."+d);
-160.....-180.....200.....10
```

(11) * Conditional operator (? :) :-

The only possible Ternary operator in Java is Conditional operator.

Syntax:

```
int x = (10<20) ? 30 : 40;
sopln(x); 30.
```

We can perform nesting of conditional operator also.

```
int x = (10>20) ? 30 : ((40>50) ? 60 : 70);
sop(x); 70.
```

a++; ++a; unary

a+b; Bi-nary

(c) ? b : d ; Ternary operator.
1 2 3 (oprand)

(12) * New operator :-
Ex: Can we new operator to create Object eg :-
Test t = new Test();

Note :- (i) After creating an object Constructor will be executed to perform initialization of object. Hence Constructor is not for creation of Object and it is for initialization of an object.

(ii) In Java we have only new keyword but not delete keyword because destruction of useless object is the responsibility of garbage collector.

(13) * [ ] operator :- We can use this operator to declare and create arrays.
eg :-
int [] x = new int [10];

In general, other than arrays we are not going to use this operator anywhere.

06/07/22 → 4:34 PM

CLASSTIME Pg. No. Date / /

Lecture 22 :- (14) Operator precedence :-

Java operator precedence chart :-

1. Unary operators:
   [ ], x++, x--
   ++x, --x, ~, !
   new, <type>

2. Arithmetic operators:
   *, /, %
   +, -

3. Shift operators:
   >>, >>>, <<

4. Comparison operators:
   <, <=, >, >=, instanceof

5. equality operators:
   ==, !=

6. Bitwise operators:
   &
   ^
   |

7. Short circuit operators:
   &&
   ||

8. Conditional operator.
   ?:

9. Assignment operators
   =, +=, -=, *= .....

(15) ★ Evaluation Order of Java Operands :-

In Java we have only operator precedence but not operand precedence. Before applying any operator all operands will be evaluated from left to Right.

eg:-

```
class Test
{
  p s v main(String[] args)
  {
    sopln(m1(1)+m2(2)*m1(3)|
    m1(4)+m1(5)*m1(6));
  }
  public static int m1(int i)
  {
    sopln(i);
    return i;
  }
}
```

O/p:-

```
3
1
2
3
4
5
6
32
```

1+2*3|4+5*6

1+6|4+5*6

1+1+5*6

1+1+30

2+30

32

Note: If multiple operator has same precedence then it execute from left to Right.

Monday Lecture 23: 6/27/22 - 28/07/22

(1) (6) new vs newInstance() :

(3) instanceof vs isInstance()

(2) ClassNotFoundException vs NoClassDefFoundError.

* new :

↳ We can use new operator to create an object, if we know classname at the beginning

Ex:
```
Test t = new Test();
Student s = new Student();
Customer c = new Customer();
```

(4) newInstance : is a method present in Class.class. We can use newInstance method to create Object. if we don't know class name at beginning and it is available dynamically at runtime from CLA, file, database.

Ex:
```
class Student
{
}
class Customer
{
}
class Test {
public static void main(String[] args) throws Exception
{ Object o = Class.forName(args[0]).newInstance();
System.out.println("object created for :" + o.getClass()
} .getName());
```

CLASSTIME Pg. No.
Date / /

Class.forName(args[0])

→ class [Class] object

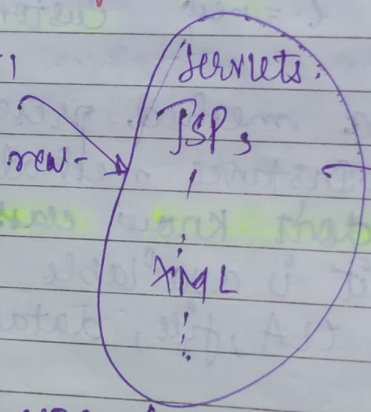
→ Op

Java Test student
Op: object created for: Student.

Java Test Customer
Op: object created for: Customer.

Java Test j.l.String
Op: object created for T.L. String.

ex2: LH /test1



Que → who is responsible to create servlet object.?
Ans → webcontainer, servlet engine., servlet container all are interchangeable. and responsible to create servlet object at runtime.

(2)

for which servlet we have to create an object webcontainer don't know until runtime.

when req come then webcontainer is going to identify url pattern in web.xml file

will search corresponding servlet and for that servlet it will create an object dynamically at runtime.

then webcontainer is used newInstance() method internally for creating an object.

Difference b/w new vs newInstance();

(1) ⇒ In the case of new operator based on our requirement we can invoke any constructor

ex: Test t = new Test();
Test t1 = new Test(10);
Test t2 = new Test("durga");

But, newInstance() methods internally called no-argument constructor hence, to use newInstance() method compulsory corresponding class should contain no-argument constructor otherwise we will get RTE: InstantiationException.

- newInstance()
  - no-arg
  - 3.

(2) While using new operator at runtime if the corresponding .class file is not available then we will get RE: NoClassDefFoundError: Test.

Eg: Test t = new Test();

At runtime if Test.class file is not available

then we will get RE: NoClassDefFoundError: Test.

While Using newInstance() method at runtime if the corresponding .class file is not available then we will get RE: ClassNotFoundException: Test123

ex:

```
Object o = (Class.forName(args[0]).newInstance());
```

Java Test Test123.

At runtime if Test123.class file is not available then we will get. RE: ClassNotFoundException: Test123.

Differences b/w new and newInstance()

new	newInstance().
① It is operator in Java.	① It is a method present in Java.lang.Class.
② we can use new operator to create an object if we know class name at the beginning.	② we can use this method to create an object if we don't know class name at the beginning and it is available dynamically at runtime.
③ To use new operator class not	

required to contain no-arg constructor.

④ At runtime if the corresponding .class file not available then we get.
RE: NoClassDefFoundError which is unchecked.

③ To use newInstance() Compulsory class should contain no-arg constructor otherwise we will get RE: InstantiationException

④ At Runtime if the corresponding .class file not available then we will get RE: ClassNotFoundException, checked.

⑩

② Difference B/w :- ClassNotFoundException Vs NoClassDefFoundError.

Imp

ClassNotFoundException Vs NoClassDefFoundError.

Ex:-
Test t = new Test();

Ex:- Test

RE: NoClassDefFoundError: Test

Ex:-
Object o = Class.forName(arg[0]).newInstance();

Java Test Student

RE: ClassNotFoundException: Student

↑ For hard-coded class names, At runtime if the corresponding .class file is not available then we will get RE: NoClassDefFoundError., which is unchecked.

For dynamically provided class name at Runtime, if the corresponding .class file is not available then we will get RE: ClassNotFoundException; which is checked exception.
At Runtime if Student.class file is not available then we will get RTE.

Que: instanceof Keyword/operator. vs isInstance(). method.

instanceof is an operator in Java we can use instanceof to check whether the given object is of particular type or not. and we know the type at the beggining.

Ex:
```
Thread t = new Thread();
sop(t instanceof Runnable);
sop(t instanceof Object);
```

isInstance() is a method present in Java.lang.Class.
We can use isInstance() to check whether the given object is of particular type or not. and we don't know the type at the beginning. and it is available dynamically at runtime.

Ex:
```
class Test
{
```

```
p s v main(String[] args) throws Exception:
{
    Thread t = new Thread();
    Sopln ( class.forName(arg[0]).isInstance(t));
}
```

→ Java Test Runable ↵
Op: true.

→ Java Test String ↵
Op: false.

isInstance() is method equivalent of instance of operator.